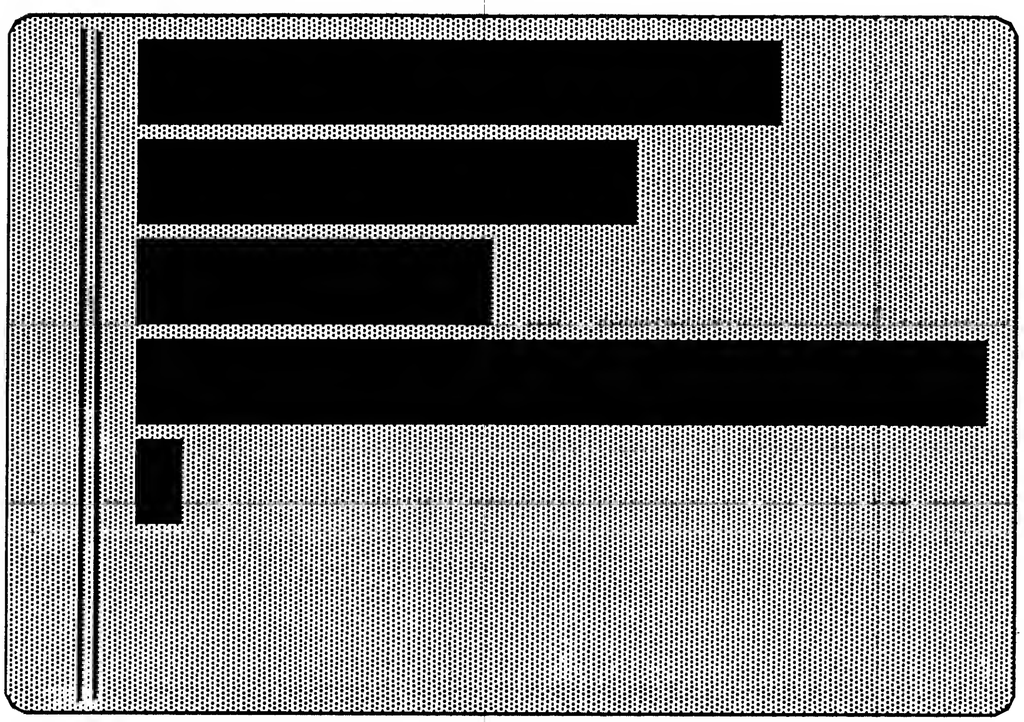
ORION INSTRUMENTS



PROGRAM PERFORMANCE ANALYZER

ORION Instruments

Program Performance Analyzer

ORION Instruments, Incorporated
702 Marshall Street
Redwood City, California
94063

The UniLab II Program Performance Analyzer

Introduction

This manual describes the Program Performance Analyzer Option (PPA), a new hardware/software tool which allows you to examine and test your microprocessor code as it executes in real time on your target board. The PPA, based on the successful UniLab II^{tm} hardware, gathers data about your program while your target board runs at full speed.

This tool not only helps you develop bug-free code, but also gives you the information you need to optimize the performance of that code.

The information in this chapter explains how to install and use the Program Performance Analyzer Option (PPA). A guide to analysis is included, together with typical examples and a troubleshooting guide.

Contents

1. Overview of the Program Performance Analyzer (PPA)

1.1 Basics

What the PPA does

The Program Performance Analyzer (PPA) gathers data about your software and displays that data as both a numeric table and a bar graph.

Confidence in your data

The data reflects the true performance of your software, because the PPA gathers performance data while your microprocessor board executes at full speed.

Menu or command interface

You gain access to the PPA features either through the commands or through the PPA menu. See section 1.5 for details.

Address-domain and Time-domain

Use the Address-domain Analyzer (AHIST) to determine the level of program activity in the entire program, or in any specified sub-section, down to a single byte.

Use the Time-domain Analyzer (THIST) to determine the run time of any single range of code. Your results will be accurate to within 20 milliseconds.

Multiple pass Address-domain

Use the Multiple pass Address-domain Analyzer (MHIST) to determine the execution time of any group of code ranges. MHIST gives you more accurate results than AHIST and expresses the results in milliseconds rather than in number of accesses to memory. The times will be accurate to within 20 milliseconds.

See section 1.3, **When to use which**, for information on choosing among the three PPA commands.

Symbolic labels

It is easy to specify the code you are interested in. All you do is enter the symbolic names of your procedures and functions. The PPA automatically converts the symbols into addresses. Or, if you want, you can manually enter the addresses.

Save data and picture

Whether you use symbols or numbers, you only have to enter the information once-- the PPA software includes a utility for saving the set-up data. You can save the data from a test to show a colleague or to include in a presentation.

To make certain that you don't mix up your files, the PPA provides a full-screen width title-- ample room for including date and program information on the screen.

You can also use the graphs in written reports. The touch of a function key (**ALT-F9**) saves the image of the screen to a text file, which you can then print out or place in a document.

A complete solution

With the PPA you can solve the problems once you find them. The underlying UniLab software provides you with all the tools you need to trace execution of your code, examine registers and memory-- even alter the code with a line-by-line assembler.

The PPA and your software's profile

The PPA collects data about the behavior of a program as it executes. You then compare the actual behavior of your program with the expected profile or "signature" of the program. You can locate problems in your code by noting the differences from your design expectations. And you can optimize your code by watching its behavior during actual running conditions.

In fact, you can alter the input conditions and watch how your program responds-- or compare two test runs gathered under two different input conditions.

1.2 Setting up the PPA

The equipment requirements

You need a UniLab II^{tm} system, version 3.12 or above. Your host computer must be an IBM compatible, with 320K or more of RAM.

How you configure your software for the PPA

The PPA is called from within the UniLab II software environment. You configure your UniLab software by using the enabling word **SOFT**. You must also copy the `HIST.OVL` file into your `ORION` directory.

The installation procedure is covered in detail in section 2.

1.3 When to use which-- choosing the correct PPA mode

You call up the PPA with one of three commands: **AHIST**, **THIST**, and **MHIST**. Both THIST and MHIST have two data collection modes, as described in section 1.4.

Address or Time domain ?

If you are interested in the general behavior of your program, or in the relative resource use of several modules, then you should use one of the two address domain histograms, AHIST or MHIST.

If you want to know the execution time of the main loop or of any <u>single</u> sub-section of the code, then you will want to use THIST, the time domain histogram. THIST is especially useful for looking at a section of code whose execution time <u>varies</u>.

AHIST or MHIST ?

You should use either AHIST or MHIST when you want to collect data on <u>several</u> address ranges of your program. The fundamental difference between the two: for each range you specify, AHIST gives you a count of the number of times memory in the range is accessed, while MHIST tells you the elapsed time between access to the first address and access to the last address in the range.

In addition, AHIST gives you faster but less accurate results. MHIST gives you greater accuracy by collecting data on only one bin at a time. Note that the bin limits you set up are preserved when you move back and forth between AHIST and MHIST.

The two modes have a similar apppearance, but act very different. AHIST collects data on all address ranges at once, but goes through a collect-analyze cycle. So it misses any events that occur while it is analyzing.

MHIST cycles through your bins, collecting data on only one bin at a time. It first finds the mean time for each address range, then finds the number of times the first address in each range is executed. Altogether MHIST requires twice as many passes as you have bins. For each "pass" you will want to restart the operation you are interested in, as described in section 6.2.

When MHIST is needed

Use MHIST whenever you need accuracy and AHIST is not able to provide it. AHIST will give you false results whenever there is "shadowing" or "swamping."

Shadowing

Shadowing occurs when AHIST consistently sees the execution of only a small section of code-- and the rest of the code falls into the shadow of this routine.

For example, your histogram shows activity only in the section of code that occurs immediately after you leave a status loop. While AHIST is busy analyzing this data, the rest of your code executes. When AHIST looks at the bus again, your program is back in the status loop.

When you have this problem, usually one bin will show a low level of activity and the others will show none. However, sometimes several bins will show activity.

Swamping

Swamping is a closely related problem: AHIST sees only the execution of the module that occurs most of the time. When the program does execute some other module, AHIST usually is analyzing the trace buffer. When you have this problem, usually one bin will show a high level of activity. While getting swamped AHIST might sporadically catch part of the execution of modules other than the dominating one.

When to use MHIST

If the data you capture with AHIST exhibits neither shadowing nor swamping, then you can be fairly confident of your results. If you do see problems, then we recommend that you use MHIST.

MHIST assumptions

MHIST will work best if your program has certain characteristics:

1) It executes (approximately) the same operation or series of operations during each "pass" of data collection. See below and in section 6.
2) The first address and last address in each routine (as specified on the MHIST screen) is accessed only once each time the routine is called.

Getting valid results with MHIST

There are three ways to make certain that your program will perform the same series of operations during each pass. See section 6.2 for more details.

1) Have reset enabled (**RESET**) so that the program starts again with each pass. You will also need to have any inputs read by your system set up in the same configuration or sequence.

2) Have reset disabled (**RESET'**) and have your program start the operation of interest when the stimulus lines change. You simply put the stimulus cable in the PROM programmer socket, and then connect the lines to the appropriate point on your board. Just before each pass, stimulus outputs S0 through S3 stobe high and then low again, while S4 through S7 strobe low and then high again. For more information on the stimulus outputs, consult section 6.2 of this document and section 8 of chapter 6.

3) Have reset disabled and manually start the operation of interest just before each pass.

1.4 The three PPA modes

The three PPA commands, **AHIST**, **MHIST** and **THIST**, give you access to three different ways to examine your program.

AHIST: address-domain histogram

In the address-domain mode, the PPA measures program activity in each user-selected address-range bin.

The column labeled "COUNT" shows, for each address range, the number of times any byte in that range is used. The graph shows what percentage of observed activity takes place in the address range of each bin.

MHIST: multiple-pass address-domain histogram

MHIST allows you to measure the absolute execution time of each address range bin (in milliseconds).

You can display either an address-domain graph of the total execution times, or a chart of average execution time, number of times called and total execution time. You will probably want the chart display when gathering data, since it gives you a clearer idea of how MHIST gathers data.

MHIST: two ways to start

You can start MHIST in one of two modes:

the Manual loop start (**F1**) and
the Timed loop start (**ALT-F1**).

In either mode, you can stop at any time by pressing either the **ESCAPE** key or function key 10 (**F10**).

Manual loop

When you press **F1**, MHIST will determine the average execution time of the address range in the first bin. MHIST will continue this operation until you press any key. It will pause, and wait for you to press another key. Then it will determine the average execution of the second bin.

After all the average execution times have been determined, MHIST will take another set of passes, determining the number of times each bin is called. It will continue to count the number of times the routine is called until you press a key, then it will pause and wait for you to press any key before it moves on to any other bin.

Timed loop

When you press **ALT-F1** MHIST will prompt you for the length of time (in milliseconds) that you want to gather data on each bin. Then it will go through the same series of operations as the Manual loop, but without any need for you to touch the keyboard.

THIST: time-domain histogram

In the time-domain mode, the PPA measures the time spent executing a section of code.

You specify the section by pressing **F9** and entering the start and stop address.

The column labeled "COUNT" shows the number of times the duration falls into each time bin. The graph displays this information as percentages of the the total number of executions. The screen also displays the mean run-time.

THIST: Two ways to collect data

There are two ways to start the THIST PPA:
the Entry-Exit start and
the Code Range start.

Entry-Exit

When started with **F1**, the PPA records the elapsed time starting when your program accesses the first address in the range and only stopping when your program accesses the last address. Note that this is the same mode used by MHIST when it gathers average execution time data.

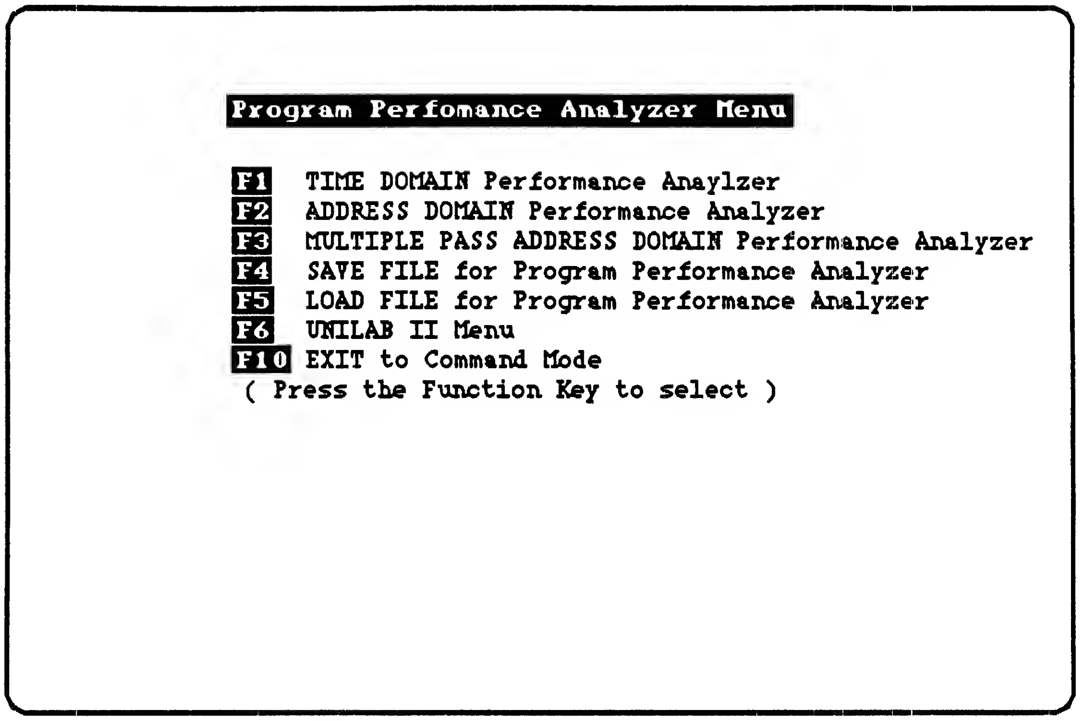
Code Range

When started with **ALT-F1**, the PPA records the elapsed time starting when the program fetches an instruction from any address within the range and stopping when the program fetches any instruction from outside the range.

> ***IMPORTANT:*** The Code Range mode will only work properly if the macro **FETCH** is defined for your processor-specific software package. Check the Glossary section of your Target Application notes.

1.5 The main PPA menu

Once the PPA is installed, a menu screen can be displayed by pressing the **ALT** key and the **F10** key at the same time. Here is what you should see when you press **ALT-F10** :



```
Program Perfomance Analyzer Menu

F1  TIME DOMAIN Performance Anaylzer
F2  ADDRESS DOMAIN Performance Analyzer
F3  MULTIPLE PASS ADDRESS DOMAIN Performance Analyzer
F4  SAVE FILE for Program Performance Analyzer
F5  LOAD FILE for Program Performance Analyzer
F6  UNILAB II Menu
F10 EXIT to Command Mode
 ( Press the Function Key to select )
```

This menu display lets you select the PPA commands by means of function keys rather than typing in the commands directly.

F1 is equivalent to **THIST**.
F2 is equivalent to **AHIST**.
F3 is equivalent to **MHIST**.
F4 is equivalent to **HSAVE** (you will be prompted for a file name).
F5 is equivalent to **HLOAD** (you will be prompted for a file name).
F6 will allow you to go directly to the UniLab II main menu mode.
This is the same as pressing **F10** from the command mode.
F10 returns you to the UniLab environment, the command mode.

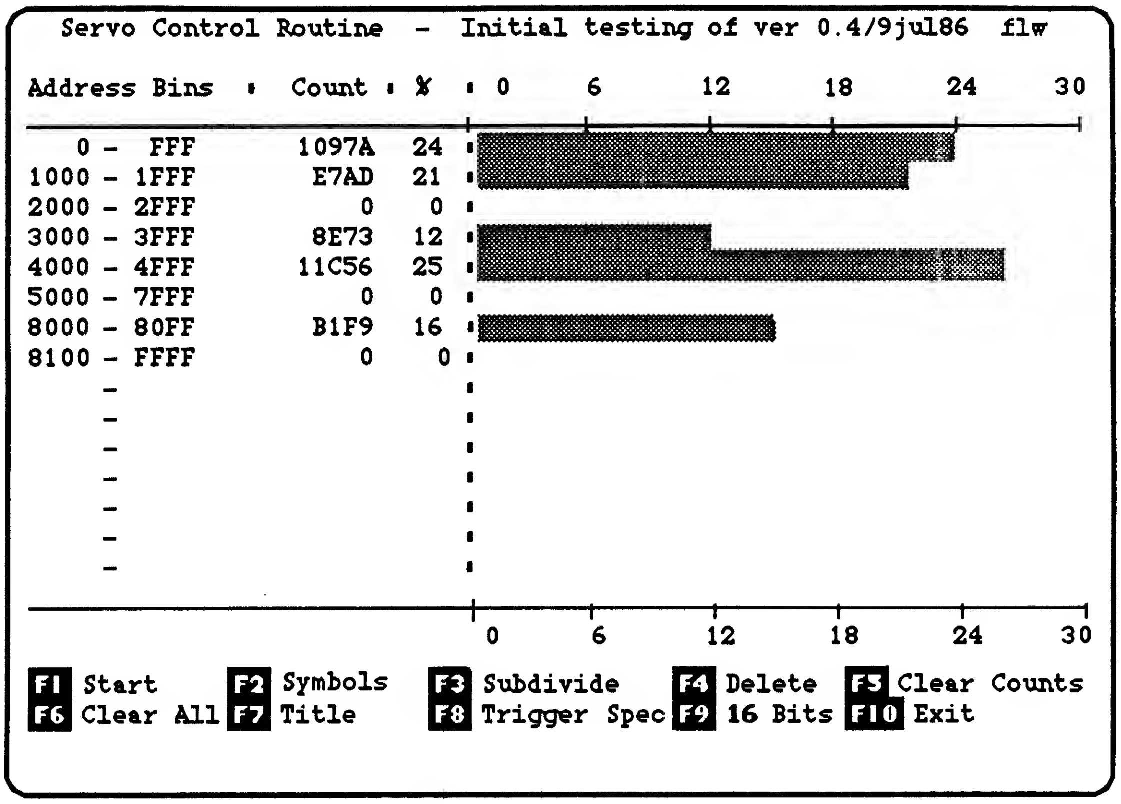
Pressing **ALT-F10** gets you back into this menu from the command mode.

In the following discussions, we will refer to the commands that you use to call up the Program Perfomance Analyzer. You may wish to use the menu in the day-to-day operation of the PPA, since you won't have to remember the command words and you won't have to type in as many keystrokes.

1.6 The interactive screen: AHIST, MHIST and THIST

All data and commands are entered into interactive screens. The screen is called up with one of the three PPA commands, **AHIST**, **MHIST** or **THIST**.

The AHIST screen display



```
 Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Address Bins  :  Count : %  : 0     6     12     18     24     30
   0 -  FFF      1097A  24 :
1000 - 1FFF       E7AD  21 :
2000 - 2FFF          0   0 :
3000 - 3FFF       8E73  12 :
4000 - 4FFF      11C56  25 :
5000 - 7FFF          0   0 :
8000 - 80FF       B1F9  16 :
8100 - FFFF          0   0 :
     -                     :
     -                     :
     -                     :
     -                     :
     -                     :
     -                     :
     -                     :
                             0     6     12     18     24     30
F1 Start      F2 Symbols  F3 Subdivide    F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title    F8 Trigger Spec F9 16 Bits  F10 Exit
```

The top line

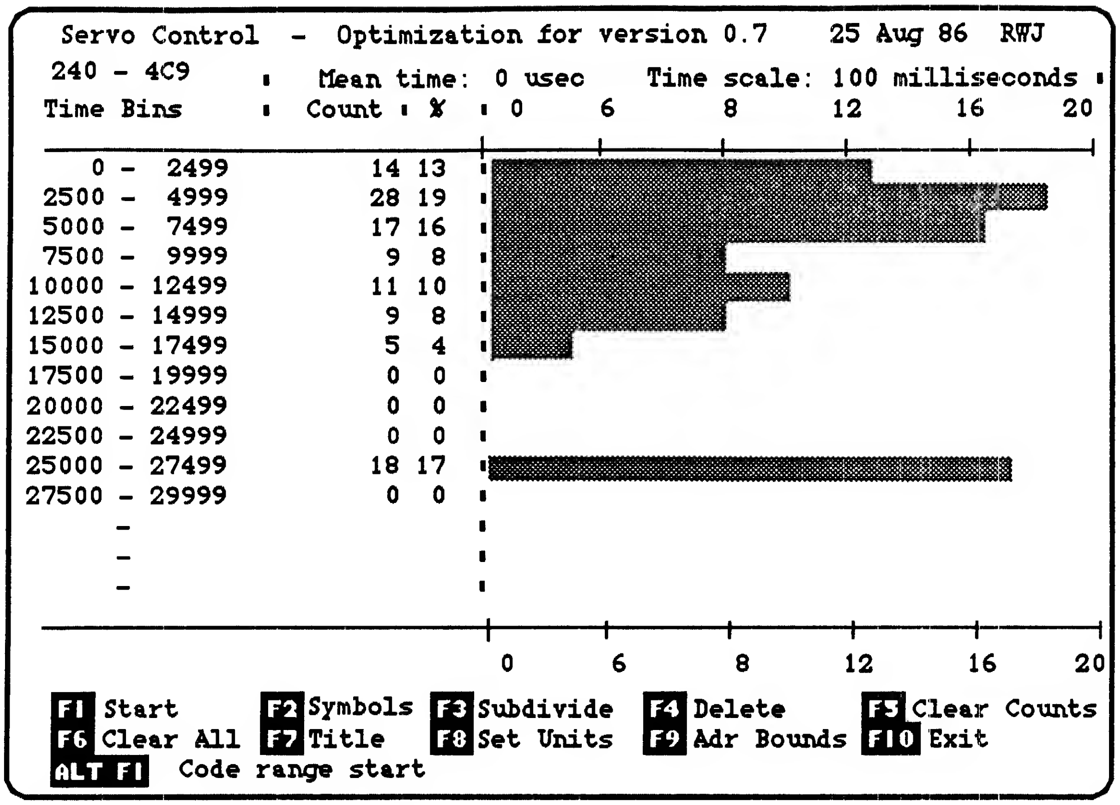
The top line of the screen is reserved for 80 characters of text. You can enter a title or notes into this field, after pressing **F7**.

The second line

The second line is blank in AHIST, but shows the address bounds, the mean time and the time scale in THIST. In MHIST it shows the time scale (currently unchangeable) and the name "`Multiple Pass Histogram.`"

You set the value of the address bounds for THIST by pressing **F9** and entering hexadecimal values. You can also enter a symbolic label-- see the discussion on the bins and symbolic labels, on page 7.

The THIST screen display



```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
240 - 4C9      |  Mean time: 0 usec    Time scale: 100 milliseconds |
Time Bins      |  Count | %  | 0     6        8       12      16     20

   0 -  2499        14 13 |
2500 -  4999        28 19 |
5000 -  7499        17 16 |
7500 -  9999         9  8 |
10000 - 12499       11 10 |
12500 - 14999        9  8 |
15000 - 17499        5  4 |
17500 - 19999        0  0 |
20000 - 22499        0  0 |
22500 - 24999        0  0 |
25000 - 27499       18 17 |
27500 - 29999        0  0 |
      -                   |
      -                   |
      -                   |

                            0     6        8       12      16     20
F1 Start      F2 Symbols  F3 Subdivide  F4 Delete     F5 Clear Counts
F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds F10 Exit
ALT F1  Code range start
```

Column titles

The third line contains the scale of the graph and the column titles.

The graph area

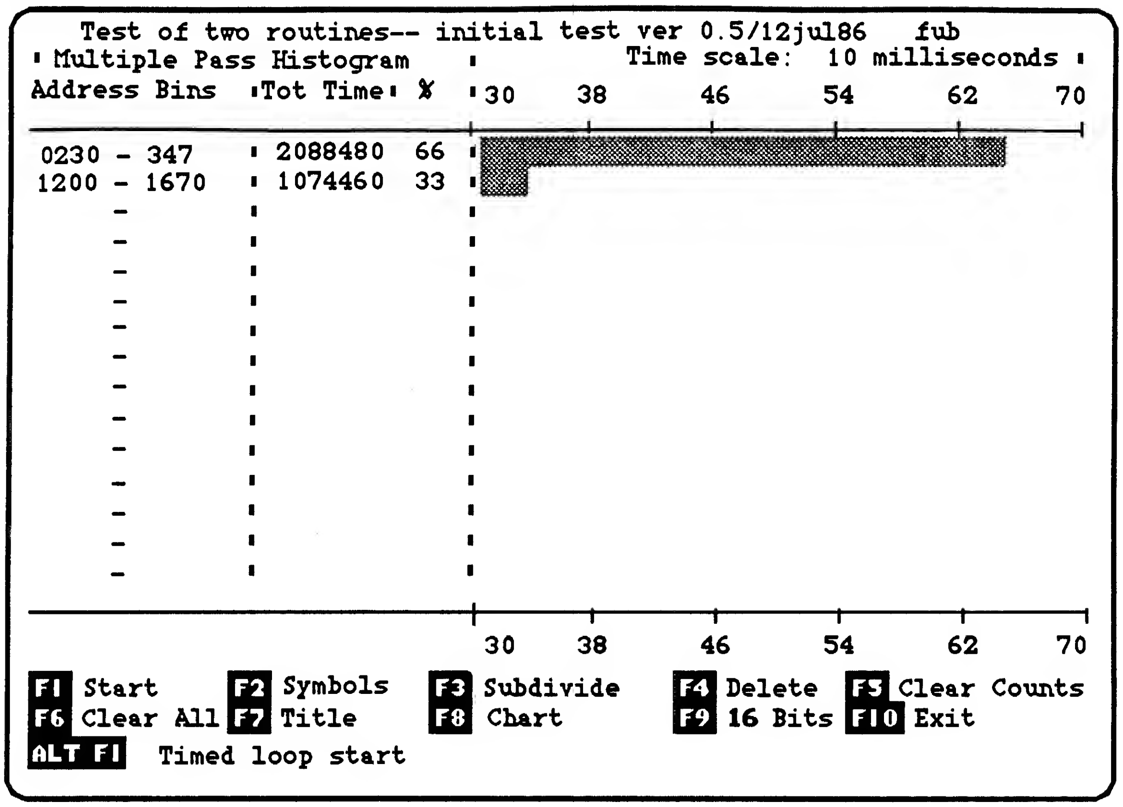
The most active portion of the screen shows a bar graph (histogram) of your program's execution. Each bar represents the proportion of activity or percentage of run-times that fall into the corresponding bin.

The scale across the top varies automatically as the data changes. It automatically adjusts the scale and the display to make the best use of the screen space.

Count and % Columns

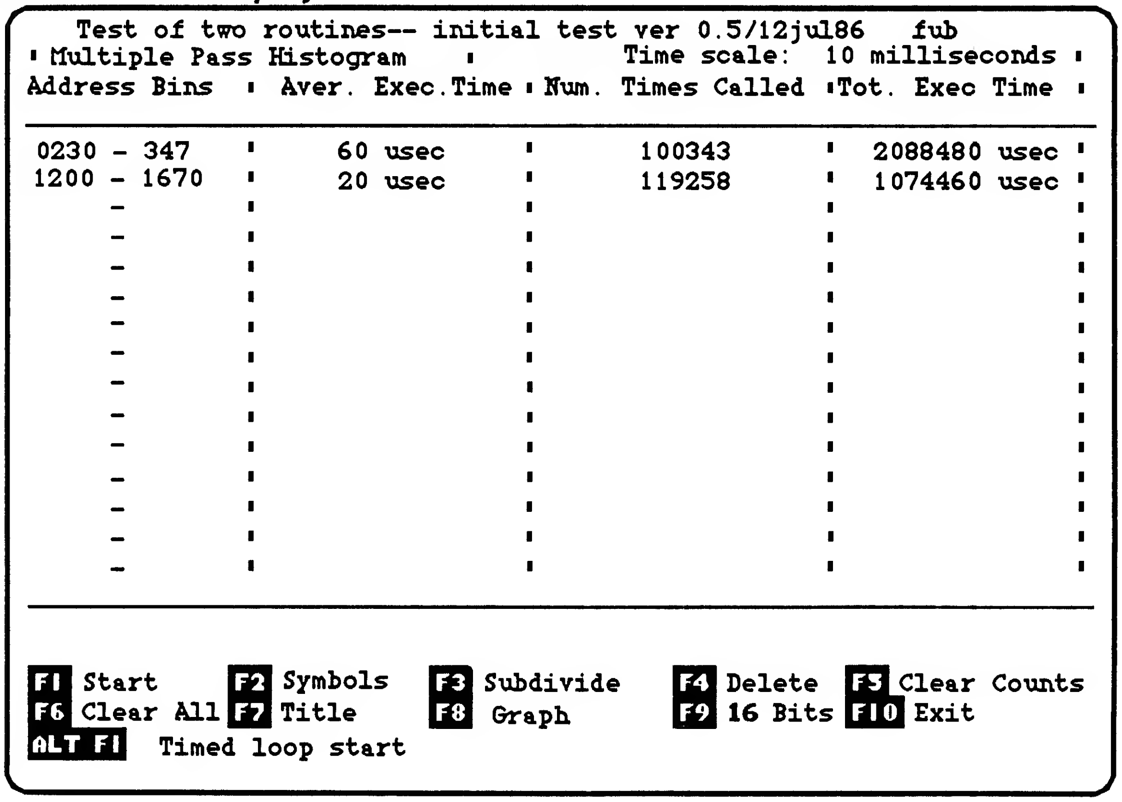
The information expressed in the graph is also displayed numerically. The two columns immediately to the left of the graph show the raw counts and the percentages.

MHIST graph display



```
   Test of two routines-- initial test ver 0.5/12jul86   fub
 Multiple Pass Histogram   |          Time scale:  10 milliseconds |
Address Bins  |Tot Time| %  |30    38     46     54     62     70
------------------------------+------------------------------------
 0230 - 347   | 2088480  66 |▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
 1200 - 1670  | 1074460  33 |▒▒▒
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
        -     |             |
------------------------------+------------------------------------
                              30    38     46     54     62     70

F1 Start      F2 Symbols  F3 Subdivide   F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title    F8 Chart       F9 16 Bits  F10 Exit
ALT F1   Timed loop start
```

MHIST chart display



```
   Test of two routines-- initial test ver 0.5/12jul86   fub
 Multiple Pass Histogram   |          Time scale:  10 milliseconds |
Address Bins  | Aver. Exec.Time | Num. Times Called |Tot. Exec Time |
------------------------------------------------------------------------
 0230 - 347   |    60 usec      |      100343       |  2088480 usec |
 1200 - 1670  |    20 usec      |      119258       |  1074460 usec |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
        -     |                 |                   |               |
------------------------------------------------------------------------

F1 Start      F2 Symbols  F3 Subdivide   F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title    F8 Graph       F9 16 Bits  F10 Exit
ALT F1   Timed loop start
```

Function key menu

The menu of commands appears below the graph area. These commands are operated by the function keys, **F1** to **F10**.

Except for **F8** and **F9**, all of the functions are the same for both AHIST and THIST.

The message areas

The two lines at the bottom of the screen are used for error messages and other miscellaneous purposes.

The line immediately above the function key menu is used only for status messages. AHIST and THIST will display only the "`Now collecting data`" message, while MHIST will display one of four messages, depending on the status of the data collection process:

```
Collecting average execution times.
Collecting number of times called.
Paused: press any key to continue.
Paused: will resume in a moment.
```

1.7 The PPA and Symbolic Labels

On the left side of the screen you enter for each bin either a symbolic label or two numbers. Press **F2** to toggle between entering numbers and entering symbolic labels.

In AHIST and MHIST a bin is a range of memory addresses, such as 10 to A0. In THIST a bin is a range of time, such as 10 to 20 milliseconds. As the PPA collects information about your program, it sorts the data into your bins.

AHIST and MHIST addresses are always entered in hexadecimal format. THIST time values are always entered in decimal, while the address is entered in hexadecimal. Time values are always output in decimal.

You can label each bin with a name, up to 14 characters long. When you enter a label the PPA will look in the symbol table for a symbol with the same name. The value of the symbol will be used as the lower limit of that bin.

If the PPA finds the symbol, then it will look for a second symbol, the same as the first except for an with "x" as a suffix. The PPA will use the value of this symbol as the upper limit, the eXit point.

For example, you can label the third bin with the name `FUNC1`. The PPA will use the value of the symbol `FUNC1` as the lower limit, and the value of the symbol `FUNC1X` as the upper limit. Of course, if you have not defined the symbols or have not loaded the symbol file into the UniLab system, then the PPA will not translate them.

See section 1.5 of chapter 6 in the UniLab Reference Manual for detailed information on loading symbol files.

You can turn off this symbol translation feature by disabling the UniLab's symbol table, with **SYMB'.**

Press **F2** to toggle between the display of bin limits and the display of the bin name.

2. Installing the Software

2.1 General

Requirements

This software package runs on the IBM PC/XT/AT^{tm} and on any computer compatible with the IBM microcomputers. Your computer must have at least 320K of RAM.

In addition, your target board must be connected to the UniLab and the UniLab connected to your host computer, as described in Chapter 2 of the UniLab Manual.

Where to go

If the UniLab software is already installed as described in Chapter 2 of the UniLab Manual, skip to Section 2.5.

If you have not already installed the UniLab software, a short installation procedure is given in 2.2 for a computer with a hard disk, and 2.3 for a computer with two floppy diskette drives. Then follow the rebooting procedure given in 2.4 before turning to Section 2.5.

2.2 Installing the UniLab System Software on a Computer with a Hard Disk Drive

Insert the master UniLab diskette into floppy drive `A:`. To execute the INSTALL batch file, enter:

```
A>INSTALL
```

The INSTALL batch file copies all of the software into the `C:\ORION` directory, and either creates or alters the `AUTOEXEC.BAT` file in the root directory.

In addition, you must create or alter the `CONFIG.SYS` file in your root directory `(C:\)` and copy the files from the Glossary diskette to the `C:\ORION` directory. There is a sample `CONFIG.SYS` file on the distribution diskette.

Refer to the UniLab Manual, Chapter 2, Software Installation, for more details.

2.3 Installing the UniLab System Software onto a PC with a Floppy Disk Drive

Put a DOS master diskette in drive `A:`. Put a blank diskette in drive `B:` and format it as a "system" diskette with the DOS command:

```
FORMAT B:/S
```

When the diskette is formatted, take your DOS master out of drive `A:` and replace it with the UniLab distribution diskette. Copy all of the UniLab files to the newly formatted diskette, using the DOS command:

```
COPY A:*.* B:
```

Now put the UniLab distribution diskette away, and put the newly configured disk in drive `A:`. Reboot your computer. Note that you must boot the computer with the correct `CONFIG.SYS` file before using the UniLab software.

Refer to the UniLab Manual, Chapter 2, Software Installation, for more details.

2.4 Rebooting the Computer

The new settings in the `CONFIG.SYS` and `AUTOEXEC.BAT` files will not take effect until the computer is rebooted.

Rebooting Procedure

With a Hard Disk:

Hold down the **CTRL** and **ALT** keys, and tap the **DEL** key.

OR

Turn the power off and back on again. (On some computers, you must wait for up to 30 seconds before turning on again.)

With a Floppy Disk:

Put the new bootable diskette in drive `A:`, then reboot as for the hard disk above.

2.5 Installing the Program Performance Analyzer Software (after UniLab Software is installed)

General

This procedure applies to both hard and floppy disk drive installations.

Procedure

1. Copy the PPA overlay file to the UniLab Diskette or to the ORION directory. The overlay filename is `HIST.OVL`.

2. Call up the UniLab program:

   ```
   ULxx.COM
   ```

   The UniLab log-on screen is displayed.

3. Enter **SOFT <CR>**.

   You are prompted to assign a filename to the PPA.

   Enter **HIST.COM** (or another filename of your choice).

   The UniLab system creates a new .COM file and then returns you to DOS.

4. Enter your new PPA **<filename>** at the DOS prompt.

   The UniLab log-on screen is displayed. It now includes the message `"Program Performance Analyzer Installed."`

This completes the software installation. Your new `.COM` file, created in step 3 above, now resides on the disk. You won't need to use the configuring word **SOFT** again.

3. Loading the Target Program into Memory

3.1 Program Source

The PPA analyzes a program running on your microprocessor control board. The program may reside in emulation memory or in a memory chip on your board.

Before you can analyze the program, you must either:

enable emulation ROM and load the program,

OR

disable emulation ROM and put your PROM chip on your board.

Program in emulation memory

Normally you will load your program into emulation memory, from either a disk file or a ROM or EPROM.

In either case you first enable emulation memory with the UniLab command

<start address> **TO** <end address> **EMENABLE**.

The address range for this command will depend on the memory map of the particular control board you are using. Refer to the UniLab Reference Manual, Chapter 6, Section 2, for a more detailed discussion. See also the entry for **EMENABLE** in chapter 7, Commands.

Program in ROM or EPROM

For final testing, you may wish to run your program directly from a memory chip on your target board.

In this case, first use the **EMCLR** command to disable emulation memory.

3.2 Loading from Disk Files

Most people will use one of two commands to load a program from a file to emulation ROM, depending on the format of the file produced by your assembler or compiler:

1. Binary format. Load the file with:

   <from addr> <to addr> **BINLOAD** <filename>

   The filenames usually end in `.BIN`, `.COM`, or `.TSK`. The screen prompts for the filename if it is not included on the command line.

2. Intel^tm-format HEX object format. Load the file with:

   HEXLOAD <filename>

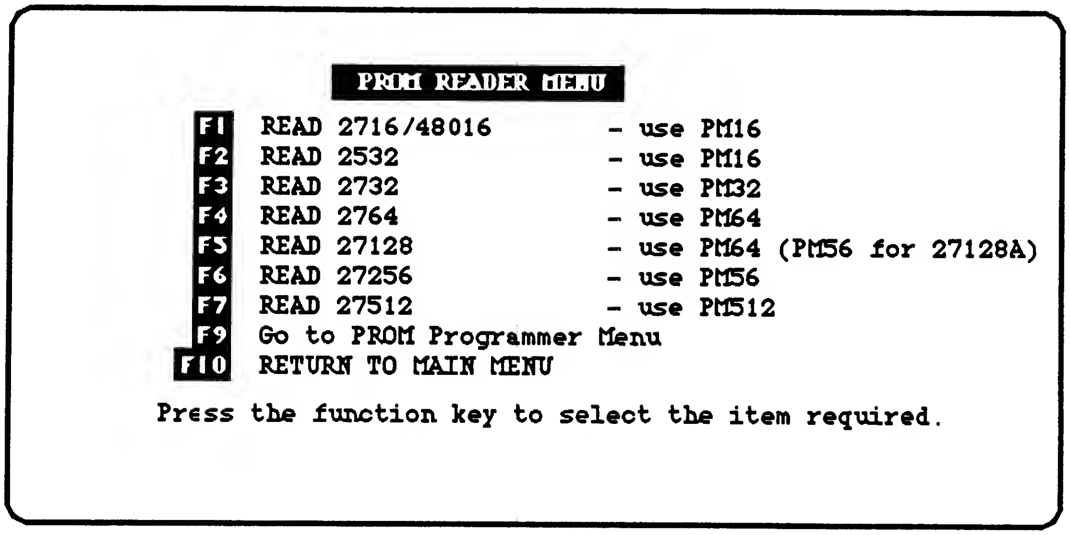
   You are prompted for the filename if it is not included on the command line. You do not specify a loading address with **HEXLOAD**.

The UniLab system also supports **HEXRCV** and **MLOADN** commands, for, respectively, loading from a remote system and loading from host RAM. Refer to the UniLab Reference Manual, Chapter 6, Section 2, on Readying and Loading Memory, as well as to Chapter 7, Commands.

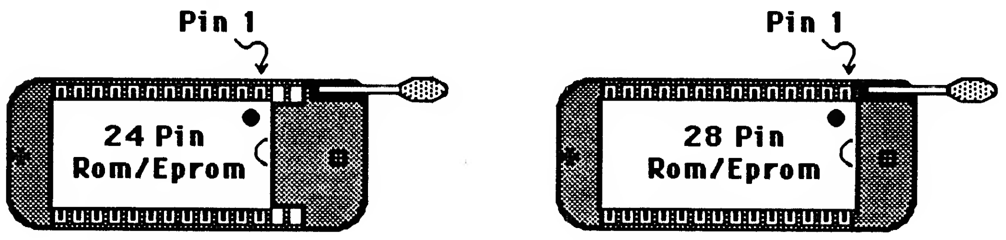
3.3 Read a Program from ROM or EPROM

You can also use the UniLab software to read a program into emulation memory from a ROM or EPROM. UniLab supports all popular devices (refer to Appendix G in the UniLab Reference Manual).

To read a program from a ROM or EPROM, first exit from the PPA screen to the UniLab environment. Then place the chip into the UniLab's prom socket, as shown below. Press **F10** for the main menu, then press **F9** to get the PROM reader menu.



```
                PROM READER MENU
      F1  READ 2716/48016        - use PM16
      F2  READ 2532              - use PM16
      F3  READ 2732              - use PM32
      F4  READ 2764              - use PM64
      F5  READ 27128             - use PM64 (PM56 for 27128A)
      F6  READ 27256             - use PM56
      F7  READ 27512             - use PM512
      F9  Go to PROM Programmer Menu
     F10  RETURN TO MAIN MENU
 Press the function key to select the item required.
```



Note that the 24-pin chips plug into the UniLab socket shifted all the way to the left.

3.4 Running a program from a ROM or EPROM on the Target Board

You may want to check the program as it runs in a ROM or EPROM on your control board. This will allow you to confirm that the program runs on the target board in its final form.

Enter **EMCLR**, to keep the UniLab emulation ROM off the bus. You can then use the PPA in the usual way.

4. Using the Address-Domain Analyzer

4.1 Summary of a Simple Procedure for Address-Domain Analysis

The program should already be in emulation ROM, or in a ROM or EPROM chip on your board, as described in section 3 of this chapter.

1. Enter the command **AHIST**, to bring up the blank AHIST screen.

2. Enter the desired address limits (in hexadecimal) into the two left columns or press **F2** and enter symbolic labels for each bin.

3. Press **F1** (Start).

Your microprocessor will start executing your code, and the PPA will start collecting data about your program. Press any key to stop program execution.

4.2 The Address-Domain Histogram

The Address-Domain Histogram shows the level of program activity in each range of addresses you specify. To analyze a new program, first load the program into memory. Then call up the AHIST screen with the command **AHIST**.

Specifying address ranges: Strategy

Memory addresses can be entered in several ways. The total range of addressable memory can be entered in one bin, and the range expanded over several bins using **F3** (Subdivide) and the cursor down key on the numeric keypad. This divides the address range equally among the bins. This is the preferred method for early studies of the program.

Alternatively, you can enter the range separately for each bin. This is the way to examine specific functions, or analyze non-contiguous sections of code.

When examining a small section of code, it may be useful to assign the remainder of memory to a single bin, so that activity outside of the section under review can be monitored. This strategy can backfire-- the PPA can end up missing the data you want to see because it is spending so much time monitoring and sorting the activity in the remainder.

Changing bin limits

When the sections of interest are identified, parts of the code can be discarded using **F4** (Delete). Other parts can be expanded for closer study using **F3** (Subdivide). Bin limits can be changed by simply typing over the address entries, so that a bin retains the same label. However, if the label is a valid symbol, the value of the symbol will override the number that you type in.

Symbolic labels

You can use a symbolic label to assign bin limits. Press **F2** to toggle between the display of addresses and the display of symbols.

After you enter a name, the PPA will search the symbol table.

If that symbol name is found it will use that value to replace the lower limit of the bin.

If it finds the symbol in the symbol table, the PPA will search for a symbol to use as the value for the upper limit. It looks for the same symbol name with an additional suffix "x."

Defining labels

The convention described above allows you to use a single name to refer to a functional range of memory.

You will have to use the appropriate label to mark the beginning and end of each range of interest in your source file. For example, label the entry point of your initialization routine with the name "PGM_START," and label the exit point of that routine with the name "PGM_STARTX."

You will then need to generate a symbol file with your assembler/linker and load that symbol file into the UniLab symbol table. See section 1.5 of chapter 6 of the UniLab Reference Manual for detailed information on symbols.

You can also make use of the UniLab command **IS** to define symbols. You will find this especially useful for defining the exit points of ranges that already have symbolic names for the entry points. For example, to define an area of memory 100 (hexadecimal) bytes long that starts at the already defined label "SORTLOOP," you type:

```
SORTLOOP 100 +   IS   SORTLOOPX
```

Many high-level languages create labels only for the entry points of functions. If you are working in a high-level language, you might find that you will have to use the **IS** command to create labels for the exit points.

Saving symbols

You can save the current symbol table with **SYMSAVE**, and load it again later with **SYMLOAD**. These commands are especially useful after you have defined symbols with **IS**, or selectively deleted symbols.

Use **SYMLIST** to get an ordered list of the current symbol file. You can then delete symbols with <symbol #> **SYMDEL**.

Labels and addresses

The PPA will automatically translate a symbol into a value when you enter the AHIST screen and each time you either move the cursor through a label field or enter a new name as a symbolic label. When the PPA finds a symbol, it overwrites the current value of the bin limit with the new value that it has found. Of course, the PPA does nothing if the label you specify is not in the current symbol table.

You can turn off the translation of symbols by disabling the UniLab symbol table. Use the UniLab command **SYMB'** to turn off the translation of symbols. **SYMB** re-enables this feature-- as do **IS**, **SYMFILE**, and **SYMLOAD**.

The address bus -- 16 or 20 bits

The address bins are normally interpreted as 16-bit numbers, even if you enter a five digit hex number (the most significant digit is ignored). Press **F9** to toggle between a 20-bit and 16-bit address bus.

A 20-bit address bus will allow the PPA to differentiate between addresses in different 64K segments. For example, 30000 and 40000 (hexadecimal) are both interpreted as the same address (0000) when you specify 16-bits, but are seen as two different addresses when you specify 20-bits.

It is a good idea not to select 20 bits unless you really need it. The high four bits of the address inputs to the UniLab hardware, A16 to A19, are usually not connected and "float high" on processors with a 16-bit address bus. This means that the UniLab sees the 16-bit address 0000 as the 20-bit address F0000. In some cases, such as the Z80, these high order address lines are used to tell the UniLab more about the activity of other control lines on the target microprocessor.

Therefore, unless your processor has a 20-bit address bus, the high four bits of the address should be ignored by the Program Performance Analyzer. This is the default condition.

Even if your processor has a 20-bit address bus, you should, for convenience, keep the PPA in 16-bit mode unless you need to distinguish between addresses in 64K segments. If you need to select addresses from a program that occupies more than one 64K segment, use **F9** to toggle to 20 bits and enter the additional digit in *all* addresses to specify the 64K segment (using ranges from 00000 to FFFFF).

Naming screens

F7 (Title) opens a field one line by 80 characters across the top of the screen. A title for the screen can be entered here, or notes to identify the screen should it be stored and retrieved at a later date. See subsection 4.5, Saving Histograms.

Modifying the trigger spec

The PPA uses the underlying UniLab command language to communicate with the UniLab hardware.

Function key **F8** gives you access to this same command language. You can use this function key to alter the trigger specification (the trigger specification tells the hardware what bus cycles to collect information on). You can alter the trigger spec so that data is not collected until after some event occurs, or so that only certain types of bus cycles are collected-- only reads, or only those with a certain data byte.

F8 opens a trap door into UniLab software environment. This means that once you press **F8** you can enter any UniLab command. However, we recommend that you only use this feature to alter the trigger spec and perform other simple commands, such as turning symbols on and off.

You can filter the trace by specifying some further criteria that each cycle must meet, besides addresses. For example, limit the trigger specification to only read cycles with the command: **READ**.

You can delay the collection of data until after a "qualifying event" has ocurred, with the command **AFTER** <qualifying event>. For example, wait until address 4500 appears on the bus with the command: **AFTER 4500 ADR** .

You can safely change any aspect of the trigger except the trigger addresses, which are automatically set by the PPA. Refer to Chapter 6, Section 4, in the UniLab Reference Manual for more information on triggers.

After you press **F8**, you are back in the UniLab environment until you press <CR>. Remember that the function keys are assigned different commands in UniLab. Be aware that if UniLab commands are invoked for a purpose other than altering the trigger, then you might have to use the command **AHIST** to re-enter the PPA.

Clearing data

F5 clears acquired data, so that another program run can be performed using the same bin allocations. **F6** clears both data and bin allocations, so that you can enter new bin limits and names.

Exiting AHIST

F10 exits the AHIST screen, and returns the system to the UniLab environment.

4.3 The Function Keys

This subsection lists the names and uses of the function keys in the menu.

F1 START

Starts the collection of data. If RESET is enabled the target program restarts from the first address. This is the recommended procedure.

Press any key to stop the data collection.

F2 NAMES/ADDRESSES

Each bin can be given a symbolic label. Pressing this key toggles between the bin address range and the label. See the discussion under section 4.2 for more details.

F3 SUBDIVIDE

This key enables bins to be subdivided. Place the cursor on any bin and press **F3**, which causes `SUBDIVIDE` in the menu to be displayed in reverse video. Then move the cursor down, and press **F3** again. The initial bin range is now divided as equally as possible among the bins between the starting and ending cursor positions.

F4 DELETE

Place the cursor over a bin and press **F4**. The name and bin limits of the chosen bin are deleted, and the bins below move up to fill the space.

F5 CLEAR COUNTS

Pressing this key resets the data count to zero.

F6 CLEAR ALL

Pressing this key clears bin limits, bin names, and data count. (`Y/N`) confirmation is required.

F7 TITLE

This function allows an 80-column title or notes to be entered across the top of the screen.

F8 TRIGGER SPEC

This key opens a "trap door" to the UniLab environment, to enable the trigger specification to be modified.

CAUTION: All function keys revert to UniLab functions until <CR> is pressed. Do not press any function key until you have typed in the command to alter the trigger spec and pressed <CR>.

F9 16 BITS/20 BITS

Toggles between 16 and 20-bit addresses. You will normally leave this set to 16 bits, especially for processors such as the Z80 that only have a 16-bit address bus.

F10 EXIT

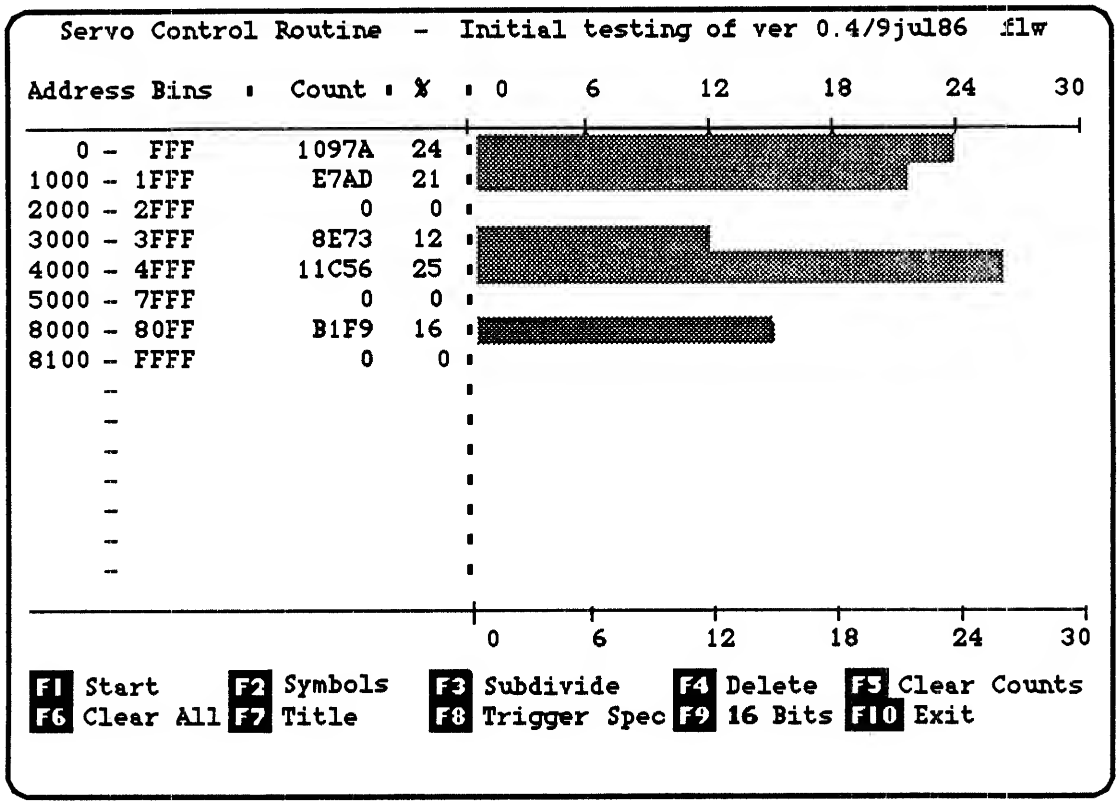
Returns you to the UniLab environment. Requires (Y/N) confirmation.

4.4 A Practical Example of AHIST

Suppose you have just completed version 0.4 of a program and you need to know how well it works. There is a main loop in ROM1 that accesses routines resident in ROM2 and ROM3. Unknown to you, there is a bug that causes a stack overflow. When this happens, the program attempts to write to ROM3 instead of to the stack. This occurs on a subroutine call, and the address on the stack tells the program where to return to. The program then tries to read back from ROM3 the address to RETURN to. At this point, the program goes berserk, accessing the wrong areas.

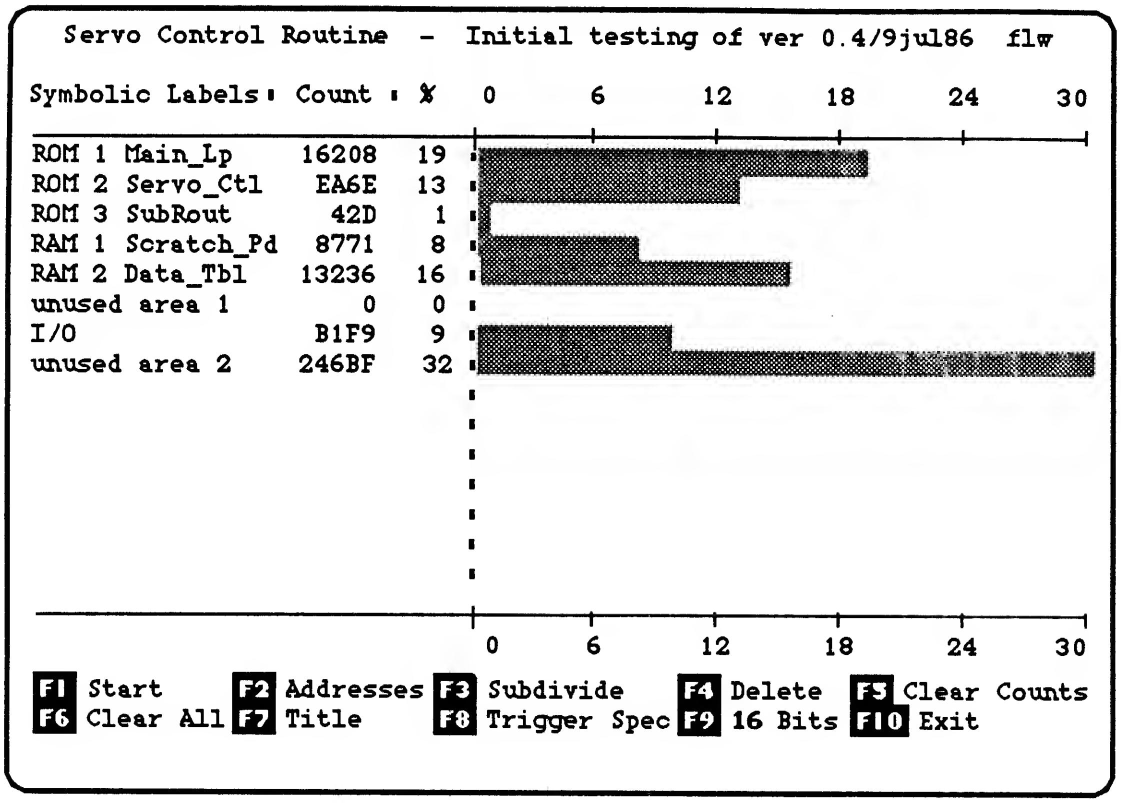
The program is run under AHIST. You type in address ranges for each chip on the board, the I/O and two unused address ranges. You use **F2** to label each bin, and press **F7** to give the screen a title.

You press **F1** to start the first run, which goes well for a time, executing the code in ROM1 and ROM2, as well as all four of the RAM chips. I/O operations also look good.



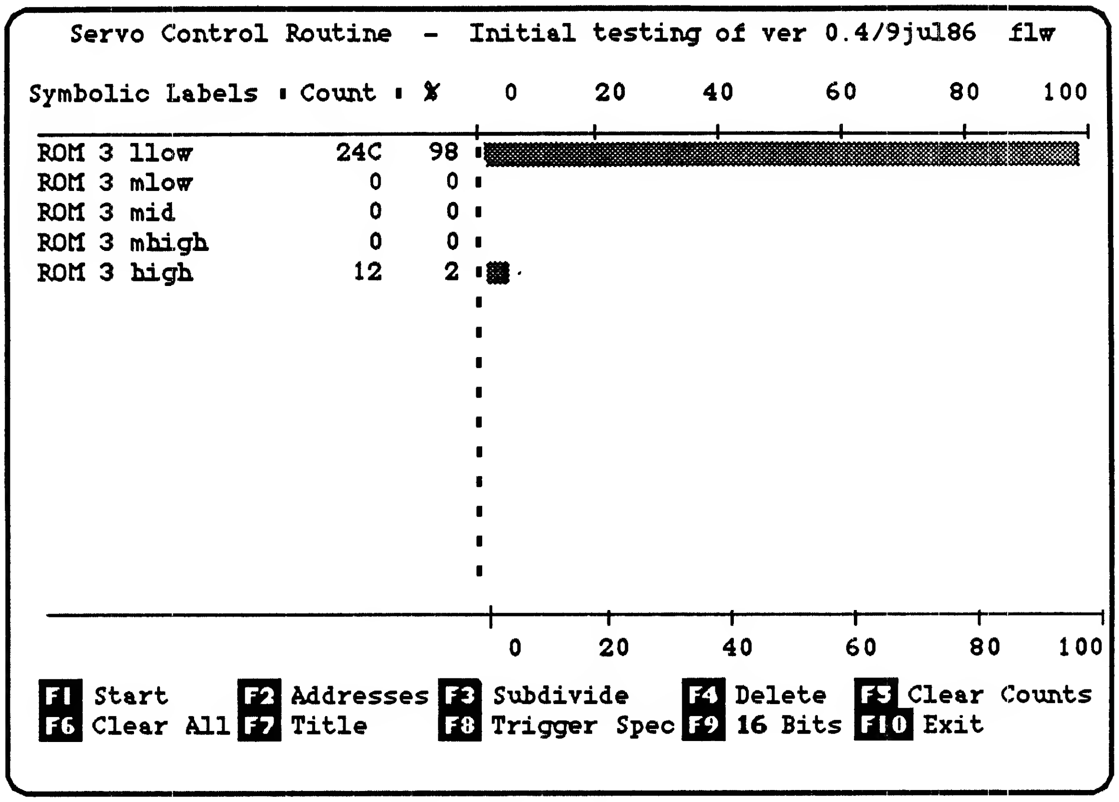
```
Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Address Bins  |  Count | %  | 0     6     12     18     24     30

   0 -  FFF     1097A   24
1000 - 1FFF      E7AD   21
2000 - 2FFF         0    0
3000 - 3FFF      8E73   12
4000 - 4FFF     11C56   25
5000 - 7FFF         0    0
8000 - 80FF      B1F9   16
8100 - FFFF         0     0
     -
     -
     -
     -
     -
     -
     -
                             0     6     12     18     24     30

F1 Start      F2 Symbols  F3 Subdivide     F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title    F8 Trigger Spec  F9 16 Bits  F10 Exit
```

Suddenly the program goes wild, running into an area of memory where it should not be. You can see that the program has reached ROM3 but not yet executed much code there. Hit any key to stop.



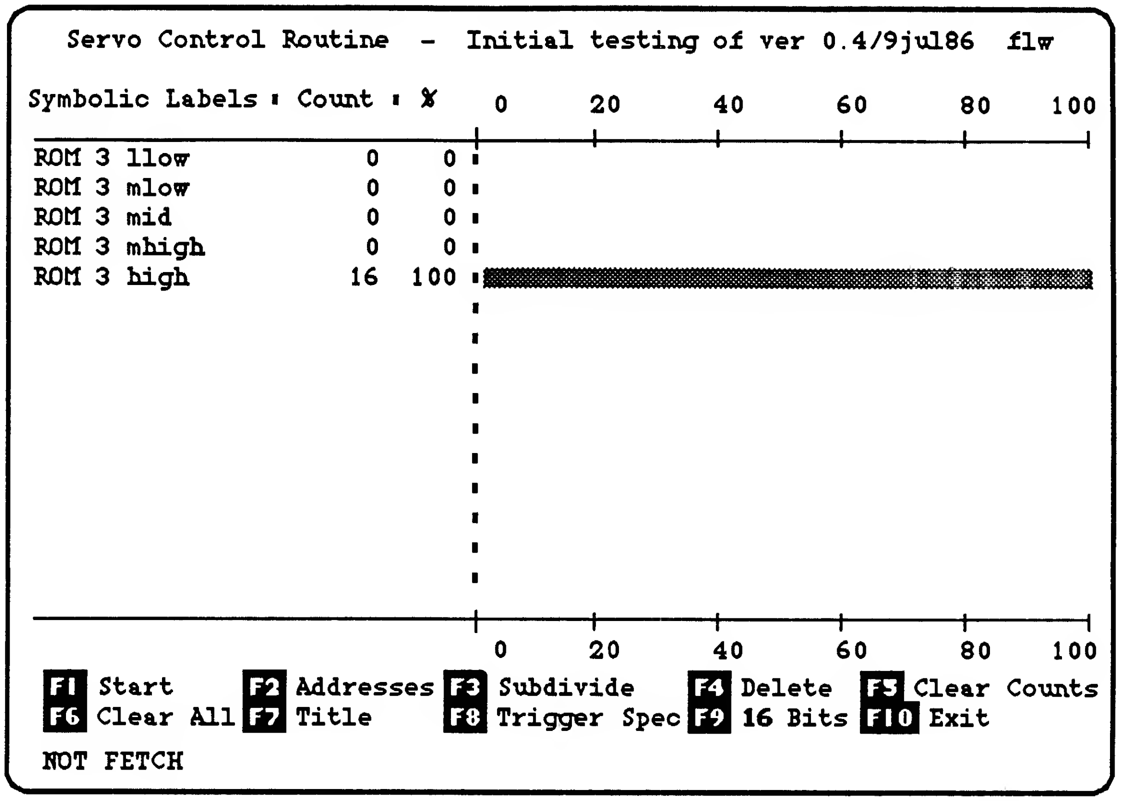
```
 Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Symbolic Labels  Count   %   0     6     12    18    24    30

ROM 1 Main_Lp     16208  19
ROM 2 Servo_Ctl    EA6E  13
ROM 3 SubRout       42D   1
RAM 1 Scratch_Pd   8771   8
RAM 2 Data_Tbl    13236  16
unused area 1         0   0
I/O                B1F9   9
unused area 2     246BF  32

                             0     6     12    18    24    30
F1 Start       F2 Addresses F3 Subdivide    F4 Delete  F5 Clear Counts
F6 Clear All   F7 Title     F8 Trigger Spec F9 16 Bits F10 Exit
```

You expand the ROM3 bin over more bins, using **F3** and the cursor down key, clear the data (**F5**) and run the program again. This time it crashes within the first section of ROM3. It mysteriously accesses the top of ROM3 twelve times.



```
  Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Symbolic Labels | Count | %    0    20    40    60    80    100

ROM 3 llow         24C   98
ROM 3 mlow           0    0
ROM 3 mid            0    0
ROM 3 mhigh          0    0
ROM 3 high          12    2

                                0    20    40    60    80    100

F1 Start      F2 Addresses F3 Subdivide     F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title     F8 Trigger Spec  F9 16 Bits  F10 Exit
```

You now suspect a stack overflow. Just to be sure, (and to demonstrate another feature), you change the trigger, pressing **F8** and entering **NOT FETCH** <CR>. The carriage return puts you back into AHIST.



```
  Servo Control Routine  -  Initial testing of ver 0.4/9jul86  flw

Symbolic Labels : Count : %    0      20      40      60      80     100
                              +-------+-------+-------+-------+-------+
ROM 3 llow          0     0 :
ROM 3 mlow          0     0 :
ROM 3 mid           0     0 :
ROM 3 mhigh         0     0 :
ROM 3 high         16   100 :▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
                            :
                              +-------+-------+-------+-------+-------+
                               0      20      40      60      80     100
F1 Start      F2 Addresses F3 Subdivide    F4 Delete   F5 Clear Counts
F6 Clear All  F7 Title     F8 Trigger Spec F9 16 Bits  F10 Exit
NOT FETCH
```

You press **F1** to begin a new test. Now the histogram shows that the access to the top of the ROM is not a fetch. Since the program seemed to be running normally in the other expected areas of memory, this is probably a bad read or write. The histogram shows no access to ROM1 or ROM2, because the processor only fetches from those chips.

At this point you can leave the PPA and look at the program using the UniLab command language. What you see there verifies your findings. With this information, you make modifications necessary for version 0.5 to run perfectly.

4.5 Saving Histograms

All the information in this section applies to THIST and MHIST as well.

There are two ways to save the graph generated by a PPA--- you can save the data, or you can save the image. If you save the data, you can later reload the same histogram and look at the data or generate a new graph with the same setup. If you save the image of the screen then you can include the image in the text of a report.

Save the data

If you wish to save the setup of a histogram as a file, you may do so with the command **HSAVE** <filename>. Issue this command after you exit from AHIST. This file includes both the setup information and the current state of the data at the time the program halted.

To re-load the saved histogram, you will use the command **HLOAD** <filename> . After HLOAD loads the data, it calls up the correct PPA mode.

Save the image

While in AHIST, press function key **F9** while holding down the **ALT** key (**ALT-F9**). You will then be prompted to type in the name of the file. When you enter a name and press the <CR> key, the image of the screen will be saved.

You can look at the image with the DOS command **TYPE**. You can also include the file in reports by importing the text file into your word processor. When you do this, you will probably have to play with the margin settings a little, to get the image to look right. The graphs use the full 80 columns of the CRT display, and you may have to edit the display, or use a compressed printing mode to print it out. The printout of the image will not show the bar graph correctly unless you have an IBM-compatible printer.

5. Using the Time Domain Analyzer

5.1 Summary Procedure for Performing a Time-Domain Analysis

The program should already be in emulation ROM, or in a ROM or EPROM on your board, as described in section 3 of this chapter.

1. Enter **THIST**, to bring up the blank THIST screen

2. Enter time limits into the time bins at the left of the screen.

3. Press **F9** and enter address bounds at the top left of the screen,
   OR press **F2** and enter a symbolic label.

4. Press **F1** to Start the program in the Entry-Exit mode,
   OR press **ALT-F1** to Start in the Code Range mode.

5.2 The Time-Domain Histogram

This feature shows you how long any specified routine takes to execute each time the routine executes. Simple routines will always take the same amount of time, unless they are interrupted.

Time bins and time scale

The collected times are sorted into the bins, for which you have specified time-ranges which bracket your guess at the actual execution time. For example, if you expect a routine to take about 10 milliseconds, then you could set the time scale to 1 millisecond (using **F8**) and subdivide a range of 0 to 30 over all the bins.

The histogram will show you the percentage of execution times that fall into each bin-range. The overall mean-time is shown at the top of the screen.

Starting THIST

There are two ways to start collecting data with the THIST PPA: Entry-Exit and Code Range. These two starts will give different results from the same setup.

The Entry-Exit start (**F1**) measures time starting when the UniLab sees the first address in the specified range. All activity, including calls and jumps to other routines, is measured until the program accesses the last address in the range.

The Code Range start (**ALT-F1**) records the time of execution from the initial fetch of any address in the code range until the program fetches an instruction from outside the range.

> ***IMPORTANT:*** The Code Range mode will only work properly if the macro **FETCH** is defined for your processor-specific software package. Check the Glossary section of your Target Application notes.

5.3 An Analogy for Understanding THIST

An analogy may help in understanding the working of the time histogram.

The investigation begins

Let's say that a woman, Rosalie, thinks that Kevin, her husband, is spending too much time in a particular tavern. She decides to get some hard data before confronting him. Rosalie develops a system where she goes to the bar every day to collect data on how long Kevin is there. When she comes home, she puts the day's reading into files: 0-2:00 hours, 2:01-4:00 hours, 4:01 to 6:00 hours, and so on. Once a week she looks through the files and updates her "bar" graph.

Subdividing

The second file starts to get fat, so she starts more files for that category, say 2:01-2:30, 2:31-3:00, 3:01-3:30, and 3:31- 4:00.

Entry-Exit

At first, when she goes to the bar, Rosalie parks outside where she can watch the entrance and exit. When she sees her husband go in the entrance, she starts her stopwatch, and lets it run until she sees him come out the exit. She then determines that he has spent 3 hours and 27 minutes in the bar.

This, of course, is the Entry-Exit method. Using this method, the unfortunate woman has no way of knowing that Kevin left the bar through the bathroom window and spent over 2 hours in the apartment across the back alley, before returning to the bar by the back way and finally leaving through the front door.

Code Range

Upon becoming aware of the situation, Rosalie changes her data-collection method. She takes up residence inside the bar (in a suitable disguise-- fake nose and glasses). She then observes whether her husband is actually present, running her stopwatch only while he is there. If Kevin leaves by the back door, the window or teleportation, Rosalie still knows how long he spent in the bar. She has, in fact, just invented the Code Range method.

The moral

The method you use depends on what you want to know. The wife in our example may in fact be more interested in how long her husband is away from home than in how long he is drinking at the bar. In this case, Entry-Exit would be perfectly appropriate.

The code-range method is better for "in-line" code-- a function or procedure that stays within a small area of memory while executing. Your code probably does not look like this, except for specialized assembly language routines.

More typically, your code will contain many jumps and branches, and most routines will call to subroutines. The entry-exit mode is more suitable for use with this type of assembly language coding. Code generated by high-level languages also tends to have this sort of structure. The entry-exit mode lets you see how much time is taken up by your routine, including any "errands" that it might run to other areas of memory.

5.4 The Time-Domain Function Keys

This is a list of the function keys displayed, together with their descriptions.

`F1` start gathering data in the Entry-Exit mode.
`ALT-F1` start gathering data in the Code Range mode.

`F2` through `F7`

These keys have the same functions as the address-domain keys.

`F8 SET LIMITS`

This sets the units of the time scale: 10 microseconds, 100 microseconds, 1 millisecond, and 10 milliseconds. For example, a bin limit value of 2 means 20 microseconds when the scale is 10 microseconds. (Note that the screen uses the abbreviation usec for microseconds.)

Use the right arrow key to change the time scale; press the **END** key in the numeric keypad to return to the THIST screen.

> NOTE: The resolution for the time histogram is accurate to within 20 microseconds. It will not be useful to set bin limit values in smaller increments.

`F9 ADR BOUNDS`

Address Bounds. These two addresses are entered in hexadecimal on the second line of the screen, or you can press **F2** and then enter a symbolic label. See the discussion of symbols in section 4.2 of this chapter for more information.

The address bounds are the starting and ending address of the code to be tested. <CR> returns the cursor to the bin limit section of the screen.

`F10 EXIT`

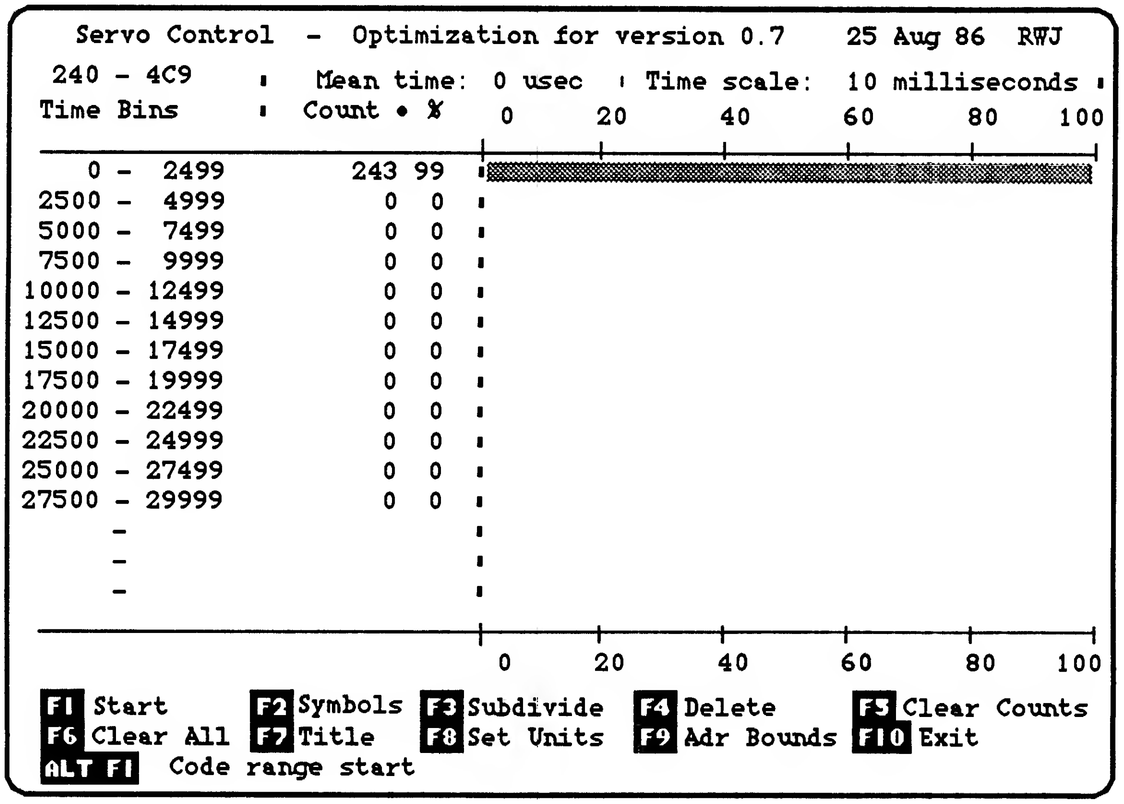
Returns you to the UniLab environment.

5.5 A Practical Example of THIST

Your program is running, and you're ready to begin optimizing the code. You decide to start by running a time test on the main loop. Unknown to you, an initialization routine is called repeatedly, and takes a long time to execute. The routine should only be called once.

You enter **THIST** and get the blank screen. You enter the address range (**F9**), using the start and stop addresses of the main loop. You set the scale to 10 milliseconds (**F8**) using the right arrow to set the time, and the **End** key (same as numeric key pad #1) to return to the THIST screen. This relatively large time scale ensures that even events that take longer than you would think are covered. You enter a time range of 0 to 30000 into the top bin, and subdivide the bin into twelve bins (press **F3**, then the down arrow 11 times to indicate a totol of 12 bins, and press **F3** again).

You press **F1** to start and run in Entry-Exit mode. All the data falls into the top bin.

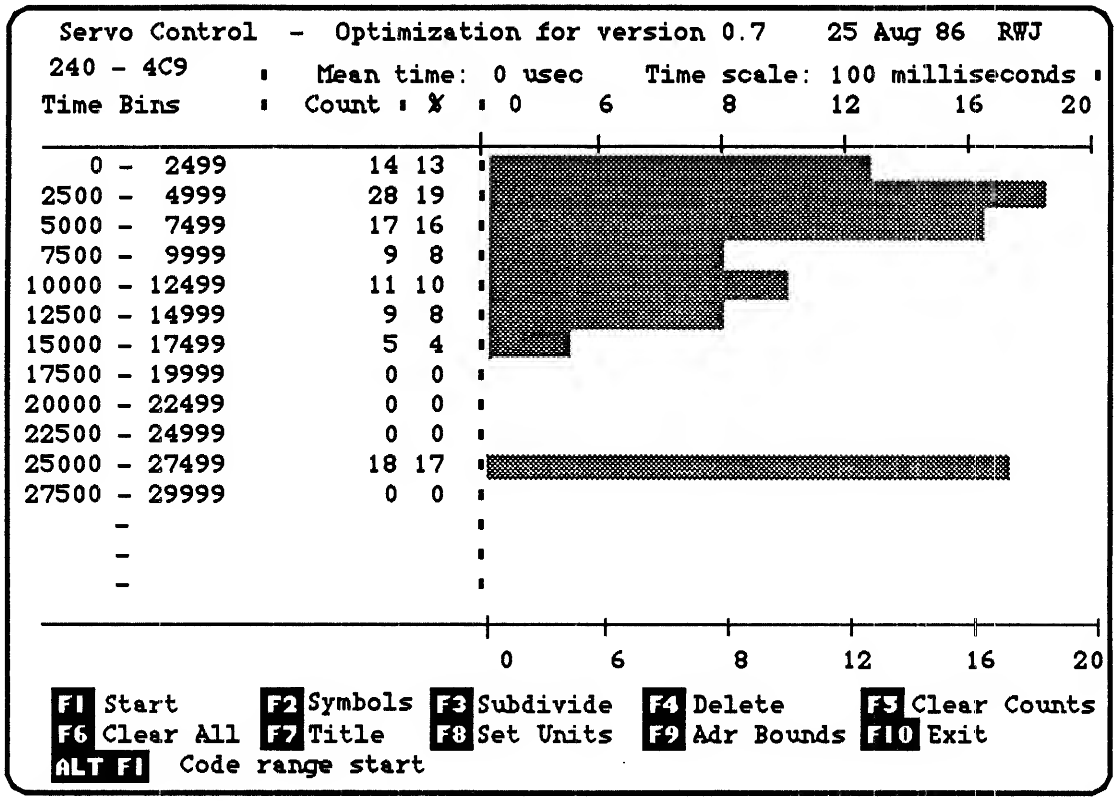


```
  Servo Control  -  Optimization for version 0.7     25 Aug 86  RWJ
 240 - 4C9      |  Mean time:  0 usec  | Time scale:  10 milliseconds |
Time Bins       |  Count • %    0      20      40      60      80     100
                                |-------+-------+-------+-------+-------|
    0 -  2499        243 99     |▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
 2500 -  4999          0  0     |
 5000 -  7499          0  0     |
 7500 -  9999          0  0     |
10000 - 12499          0  0     |
12500 - 14999          0  0     |
15000 - 17499          0  0     |
17500 - 19999          0  0     |
20000 - 22499          0  0     |
22500 - 24999          0  0     |
25000 - 27499          0  0     |
27500 - 29999          0  0     |
      -                         |
      -                         |
      -                         |
                                |-------+-------+-------+-------+-------|
                                0      20      40      60      80     100
F1 Start      F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
ALT F1   Code range start
```

Since you don't like this display, you clear the counts with **F5**.

Then you change the time scale to 100 microseconds with **F8**, and run the program again.

You now get a much better display of data. The times fall into several bins--including a surprising number in the 25000-27499 bin. You suspect that the large number of execution times indicates a problem. You decide to investigate further.



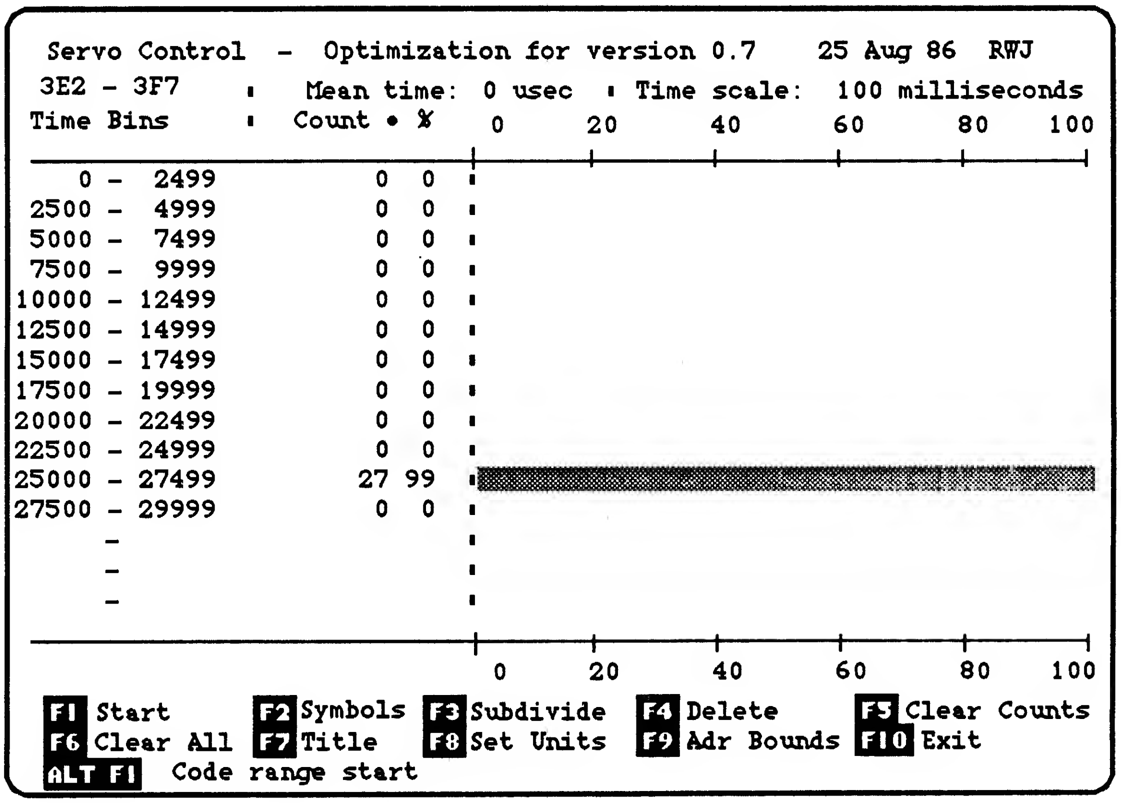
```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
 240 - 4C9       ▮  Mean time: 0 usec    Time scale: 100 milliseconds ▮
 Time Bins       ▮  Count ▮ %  ▮ 0    6      8      12     16     20
       0 -  2499      14 13
    2500 -  4999      28 19
    5000 -  7499      17 16
    7500 -  9999       9  8
   10000 - 12499      11 10
   12500 - 14999       9  8
   15000 - 17499       5  4
   17500 - 19999       0  0
   20000 - 22499       0  0
   22500 - 24999       0  0
   25000 - 27499      18 17
   27500 - 29999       0  0
         -
         -
         -
                             0    6      8      12     16     20

 F1 Start      F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
 F6 Clear All  F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
 ALT F1  Code range start
```

In order to narrow down the problem area, you clear the counts again (**F5**) and change the second address (**F9**) to point at the half-way mark of the main loop. This time when you press **F1** you don't get any long execution times.

You now know that the long execution times are caused by code that either resides in or is called from the second half of the main loop. You press **F9** again, and change both the starting and ending address, so that you are examining only the third quarter of the main loop.

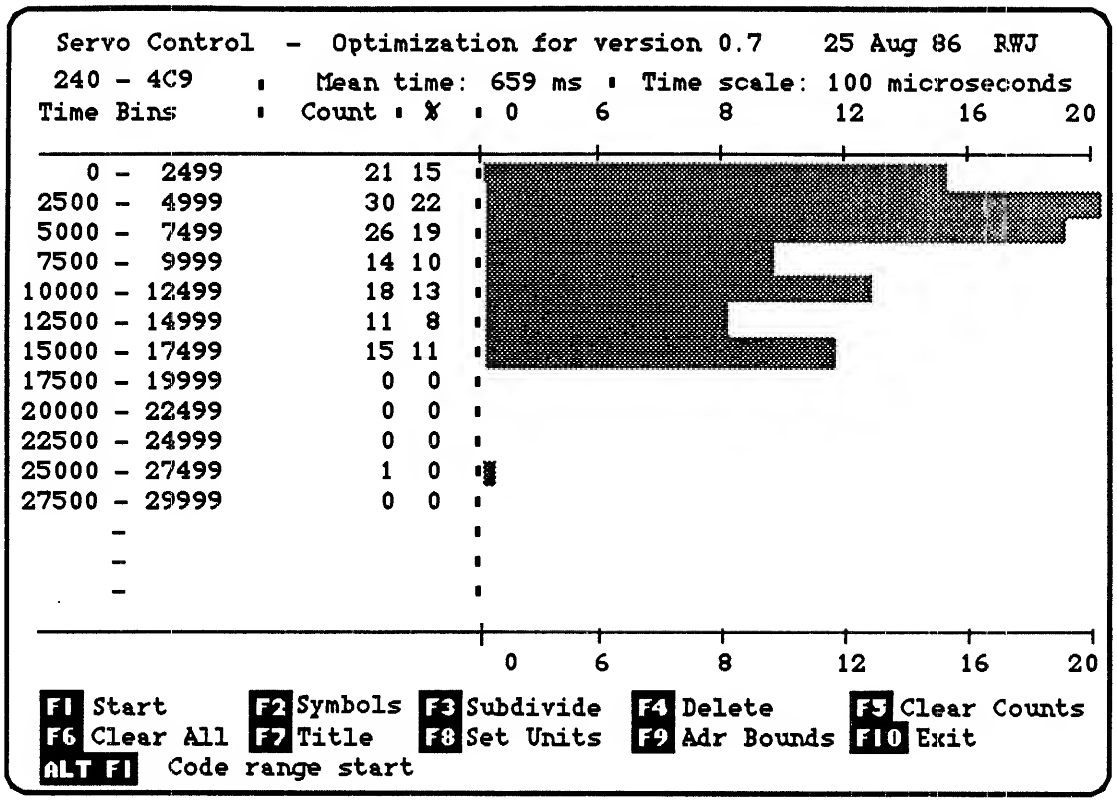
This time, the code often takes a long time to run. You look at the listing for that part of your program and find a call to an initialization routine that you suspect is delaying the program execution. You clear the counts again, and this time enter addresses just before and just after the suspiscious call .

You press **F1** once again, and immediately see that there are multiple, lengthy run times.



```
 Servo Control  -  Optimization for version 0.7    25 Aug 86  RWJ
 3E2 - 3F7     ▮  Mean time:  0 usec  ▮ Time scale:  100 milliseconds
Time Bins      ▮  Count • %     0     20     40     60     80    100
    0 -  2499          0  0  ▮
 2500 -  4999          0  0  ▮
 5000 -  7499          0  0  ▮
 7500 -  9999          0  0  ▮
10000 - 12499          0  0  ▮
12500 - 14999          0  0  ▮
15000 - 17499          0  0  ▮
17500 - 19999          0  0  ▮
20000 - 22499          0  0  ▮
22500 - 24999          0  0  ▮
25000 - 27499         27 99  ▮▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
27500 - 29999          0  0  ▮
        -                    ▮
        -                    ▮
        -                    ▮
                                0     20     40     60     80    100
 F1 Start     F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
 F6 Clear All F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
 ALT F1  Code range start
```

Now the program can be corrected, and the test of the routine rerun. You have found the bug and exterminated it. The trace of the corrected program shows, properly, only one very long execution time, indicating that the initialization routine is now called only once.



```
 Servo Control  - Optimization for version 0.7    25 Aug 86  RWJ
 240 - 4C9     ▪  Mean time: 659 ms ▪ Time scale: 100 microseconds
Time Bins      ▪ Count ▪ %  ▪ 0     6      8      12     16     20

    0 -  2499     21 15 ▪
 2500 -  4999     30 22 ▪
 5000 -  7499     26 19 ▪
 7500 -  9999     14 10 ▪
10000 - 12499     18 13 ▪
12500 - 14999     11  8 ▪
15000 - 17499     15 11 ▪
17500 - 19999      0  0 ▪
20000 - 22499      0  0 ▪
22500 - 24999      0  0 ▪
25000 - 27499      1  0 ▪
27500 - 29999      0  0 ▪
      -                 ▪
      -                 ▪
      -                 ▪

                          0     6      8      12     16     20
F1 Start       F2 Symbols  F3 Subdivide  F4 Delete      F5 Clear Counts
F6 Clear All   F7 Title    F8 Set Units  F9 Adr Bounds  F10 Exit
ALT F1   Code range start
```

5.6 Saving Histograms

See section 4.5. The commands for saving histogram data and images is the same for all three PPA modes.

6. Using the Multiple-Pass Address-Domain Analyzer

6.1 Summary of a Simple Procedure for Multiple-Pass Analysis

The program should already be in emulation ROM, or in a ROM or EPROM chip on your board, as described in section 3 of this chapter.

1. Enter the command **MHIST**, to bring up the blank MHIST chart screen (NOTE: if you enter MHIST after leaving AHIST, then the bin limits will be preserved. Within MHIST you can use **F6** to clear the bin limits).

2. Enter the desired address limits (in hexadecimal) into the two left columns or press **F2** and enter symbolic labels for each bin. Note that MHIST allows overlapping bins, or nested ones for that matter.

3. Press **F1** to Start collecting data in Manual Loop mode,
   OR
   press **ALT-F1** to Start in Timed Loop mode.

In **Manual Loop** mode, your microprocessor will start executing your code, and the PPA will determine the average execution time of the first bin. You can press any key to make the PPA pause, then when you press any key again, the PPA will collect data on the second bin. When the PPA has the average execution time of all bins, it will start collecting the number of times called for each bin. So to move from bin to bin you will need to press a key to pause and then again to continue.

In **Timed Loop** mode, you will be prompted to enter the length of time to collect data on each bin. You must enter this (decimal) number in milliseconds. The maximum is approximately 65000 milliseconds, or 65 seconds. The PPA will then determine the average execution time and the the total number of times called for each bin. You will not need to press another key after you enter a valid number and press carriage return. If you do press a key, you will prematurely stop collecting data on the current bin.

In **either** mode, you can stop at any time by pressing either **F10** or the **ESC** key. And in both modes the figures you get will be expressed in milliseconds and will be accurate to within 20 milliseconds.

6.2 The Multiple-Pass Address-Domain Histogram

See section 4.2. The information in that section also applies to MHIST, except for the discussion of specifying trigger specs. Additional information specific to MHIST appears below.

The Multiple-pass Address-Domain Histogram shows the average execution time, number of times called and total execution time of each bin. The average execution time is approximate, while the number of times called is exact. To analyze a new program, first load the program into memory. Then call up the MHIST screen with the command MHIST.

Changing from Chart to Graph

When you call MHIST, it will start up displaying the Chart screen. This display shows you the average execution time, the number of times called and the total execution time for each bin.

Press **F8** to toggle between chart display and the graph display. The graph display shows you only the total execution time and a histogram of the total execution time.

Both displays highlight the number that is being altered while gathering data.

Getting valid results: starting with RESET, manually triggering an action or using the Stimulus outputs

The results that MHIST gives you are only valid if your target software is executing the same series of instructions each time you start it up. For this reason you should use one of three strategies:

1) The simplest is to have reset enabled (**`RESET`**), which will cause the target program to start over from the beginning each time. With this method you can perform a timed loop start.

2) Disable reset (**`RESET'`**) and manually start some operation on your target system at the start of data collection for each bin. With this method you would want to perform a manual loop start.

3) Disable reset (**`RESET'`**) and use the stimulus outputs to trigger some operation on your target system at the start of data collection for each bin. MHIST automatically sends a strobe out on the stimulus outputs at the start of data collection for each bin. With this method you can use a timed loop start.

MHIST normally holds the outputs S0 through S3 low and S4 through S7 high. Just before MHIST starts collecting data on each bin, it reverses these outputs and then returns them to their usual values.

line # :	0	1	2	3	4	5	6	7
normal value:	0	0	0	0	1	1	1	1
value during strobe:	1	1	1	1	0	0	0	0

Note that MHIST sends these signals out whether or not you use them.

Additional stimulus information

You can only use the stimulus outputs if your program reads an input value and then takes an action when a certain value appears-- or if you are willing to write the extra code to test the value and take an action. The best way to use these stimulus outputs is to look for the positive-going or negative-going edge.

To use the stimulus outputs you need to connect the stimulus cable to the PROM programmer socket, as shown below. The ends of the cables are labeled.

See section 8 of chapter 6 if you need additional information on the stimulus outputs.

6.3 An analogy for MHIST

An analogy may help in understanding the multiple-pass histogram.

A suspiscion is hatched

Divorce seems imminent for Rosalie and Kevin, the couple who starred in the THIST analogy.

Kevin has become suspiscious of his wife, and has decided to keep track of how she spends her time. He decides that if would be too obvious to tail her. Instead he will watch outside of three locations he knows Rosalie frequents: her place of work, a health spa and an cappuccino bar.

Catching the average time

He spends a day in front of each place, watching when she enters and leaves. Each time she leaves he writes down the elapsed time and resets his stopwatch. That way he is able to determine how long she spends inside each place, on the average.

A data collection problem

Kevin realizes that his wife might only step outside for a breath of fresh air, and then immediately go back inside. He would not be able to spot these entrances, since he is busy writing down the elapsed time and resetting his watch. So Kevin doesn't even know if Rosalie behaves this way.

MHIST has the same difficulty-- while the PPA is recording the elapsed time and resetting the clock, your program could re-enter the routine you are monitoring. That's why you need the second pass through each bin.

Determining the number of events

Kevin, who has lost his job by now, decides to spend another three days on the project. This time, he spends a day in front of each place, just watching the entrance and keeping track of how often his wife goes in. He figures that once he knows how often Rosalie goes into each building, he can multiply the average stay by the number of stays and thus calculate the total amount of time she spends in each building.

The husband's assumptions

Kevin has made two assumptions:

1) Rosalie's behavior is constant from day to day.

2) The average time he calculated is valid, even though it might not include all of the visits Rosalie made to each location. That is, Kevin assumes that the visits he misses (when determining average time) do not deviate from the mean.

Of course, we make similar assumptions when using MHIST.

Kevin's method: stimulus and response

Kevin has cleverly manipulated Rosalie, to protect the first assumption. He knows that she will always follow the same routine after they have an argument. So on the morning of every day that he wants to gather data, he starts the same fight with her.

Of course, this couple has been having the same argument every morning for the last five years, so Rosalie suspects nothing.

The moral

Remember that your program must perform the same operation during each pass, or your results will not be valid.

As for Kevin and Rosalie: they were thinking of divorce, but then realized that no one else would be able to put up with their bizarre behavior.

6.4 The Function Keys

This subsection lists the names and uses of the function keys in the menu.

F1 START

Starts the collection of data. If RESET is enabled the target program restarts from the first address each time the PPA starts to collect data on a bin. This is the recommended procedure.

Press any key to pause, and then press any key again to continue with the next data collection.

The PPA will stop collecting data when you have looped through all the bins twice--once for average execution time and once for number of times called. Or you can press **F10** or the **ESC** key to stop the data collection

ALT-F1 TIMED LOOP START

Starts the collection of data in the timed loop mode.

F2 through **F7**

These keys have the same functions as they do under AHIST.

F8 CHART/GRAPH

This key toggles between the chart display and the graphical display of the data collected by MHIST. When you are collecting data, you will probably want the Chart display, since it gives you a clear idea of what the PPA is doing.

F9 16 BITS/20 BITS

Same as under AHIST, toggles between 16 and 20-bit addresses. You will normally leave this set to 16 bits, especially for processors such as the Z80 that only have a 16-bit address bus.

F10 EXIT

Returns you to the UniLab environment. Requires (Y/N) confirmation.

6.5 Saving Histograms

See section 4.5.

7. Troubleshooting

7.1 Operating problems

Here are some hints that will help you avoid problems.

Installation Hints

- Always move the `HIST.OVL` file into the `ORION` directory on your working disk.

- Make certain that you have entered the configuration word **SOFT** from within the UniLab software.

- Always use the correct version of the `CONFIG.SYS` file.

Target Program Loading

If the program is run in the emulation memory, the proper range of emulation ROM must be enabled (**EMENABLE**). If the program is run from the chip, the emulation ROM must be disabled (**EMCLR**).

The **RSP'** command can be used to disable the debug for a completely transparent emulation during the use of the PPA. When you do this, be sure to re-enable the debug with **RSP** before attempting to set a breakpoint.

If the debug is not disabled, it will insert code into the ORION reserved area for your processor. Refer to the UniLab Reference Manual and the Target Application notes for further information.

Symbolic labels

Use the UniLab command **SYMLIST** to verify that your symbol file has loaded properly. You can use <symbol #> **SYMDEL** to delete any unwanted symbols.

The PPA will look at all the symbolic label fields whenever you enter AHIST or THIST. If it finds a symbol then it will update the address bin. If you don't want it to do this, then disable the symbol table with **SYMB'**. You can enable the symbol table again with **SYMB**. Several other commands also re-enable the symbol table: **IS**, **SYMFILE**, and **SYMLOAD**.

The PPA will not try to find the symbol for the exit address (with suffix "X") unless it finds the symbol for the entry address.

If AHIST or MHIST Does Not Run Properly:

- check that the 16-bit address (**F9**) is selected, unless the program operates across 64K segments.

- make certain that RESET is in the state you want (either enabled or disabled).

If THIST Does Not Run Properly:

- check that the address range entered is within the range of the program under test.

- check that the starting and ending addresses are correct.

If you are getting the wrong information using **ALT-F1** (Code Range mode):

- make sure that the macro **FETCH** is defined for your processor- specific software package.

- if you have a processor that has "extra" bus cycles, as many Motorola processors do, make certain that those bus cycles do not appear to be fetches from an address outside of the normal code range.

If you get an "RS-232 error" after using the PPA

- the UniLab hardware can end up in an indeterminate state if the PPA is exited from abnormally. The command `INIT` will not work-- but all you have to do is turn the UniLab off for a second, turn it on again and then type in `INIT`.

7.2 Error Messages

`Bad Range - can't subdivide` - This error occurs if you try to subdivide a bin with an invalid or missing number, or one that has the lowbound larger than the highbound.

`Boundaries for bins overlap` - AHIST and THIST will not produce a histogram until this error is fixed. It occurs if any two ranges of addresses or times share a region. For example,

1000-2000 and 1500-2500

or even just

1000-2000 and 2000-3000.

`Can't create file` - This error occurs if the name you specify for the file is an invalid name, or if DOS cannot create a file for some other reason.

`Disk full` - This error occurs if you try to use **TSAVE** or try to save a screen image file when there is not enough room on your disk.

`File not found` - If you try to load a non-existent file with **HLOAD**, you will get this message. Common errors are mispelling, using the wrong file extension, or not specifying the proper path. You can look at the disk directory at any time by pressing **F10** to leave the PPA display and then typing

`DOS DIR` (or **`DOS DIR A:`** or any valid DOS command).

You must be in the UniLab environment (command mode) to do these DOS functions.

`Invalid or missing number` - This error occurs if you try to run a test with no bin defined, or with one limit defined but not the other, or with a label entered but no values, or with a numeric field containing a value that is not a number in the base you are using. For example, FF is not a number in decimal (which THIST requires). You will not be able to produce a histogram until you correct the mistake.

`Invalid start and stop address for THIST` - This error tells you that one of the two address bounds that you gave to THIST is missing or is not a number in the base you are using.

`Lowbound is larger than highbound` - This error occurs if a bin has a starting value that is higher than the ending value. You cannot produce a histogram until you correct the mistake.

`Not enough bins available` - This error occurs if you try to allot more than 15 bins using **F3** (Subdivide). This can occur if you already have several bins alloted and then try to subdivide one of them among all the bins. Be sure to delete enough bins to allow room for expansion.

`RS-232 error #`XX - This error can occur after an abnormal exit from the PPA software. You will have to turn off the UniLab, wait a second and turn it back on. Then you will be able to type the command **`INIT`**.

8. Specifications

8.1 Operating method and limitations

AHIST

The AHIST PPA works by collecting 170 bus cycles at a time in a trace buffer and sorting them into the bins that you specify. The PPA filters the data it gathers, so that the addresses of the 170 cycles fall between the highest and lowest address that you specify.

Generally, you can be confident of your results once the percentage data has stabilized.

However, this method has two main limitations:

1) If you are looking exclusively at a range of code that seldom occurs, you will get a "shadow" effect. The first 170 cycles of the routine will be gathered, and then the PPA will not gather data during the time it is sorting the data into bins.

2) If you are looking at a very small range of code that occurs periodically, and also looking at a range of code that occurs continuously, you can get a "swamped-out" effect. The trace buffer will be continually filling up with cycles from the code that is continuously executing, and might miss the execution of the small code range.

MHIST

The multiple-pass histogram collects average times the same way that THIST does, and then collects the number of times called by keeping a count of the number of times that the first address is accessed.

The times are accurate to within 20 milliseconds, and the count of the number of times called is accurate up to FFFF hex, provided that the first address is accessed only once each time the routine is called.

THIST

The THIST PPA works by timing the duration of a function. It sorts each sample into a bin as soon as it finds one. You can have confidence in the results to within 20 microseconds.

Glossary of commands

AHIST
Calls the address-domain histogram, described in section 4 of this chapter.

HLOAD `<filename>`
Loads the histogram data stored in the file, and then calls up the proper histogram. Use this command only on files saved with **HSAVE**.

HSAVE `<filename>`
Saves the data from the last histogram displayed, after exiting to the UniLab environment.

MHIST
Calls the multiple-pass address-domain histogram, described in section 6 of this chapter.

SSAVE
Saves the screen image as a DOS text file. This command is always assigned to function key **ALT-F9**, whether you are in the PPA or not.

THIST
Calls the time-domain histogram, described in section 5 of this chapter.